Session 4

White-Box Testing (2)

xfzhang@suda.edu.cn

Objectives

- Review of Session 3
  - Basic Concepts of White box testing
  - Logic Coverage
  - Control Flow Graph
  - Basis Path Testing
  - Loop Testing
  - Data Flow Testing

3.1 Basic Concepts

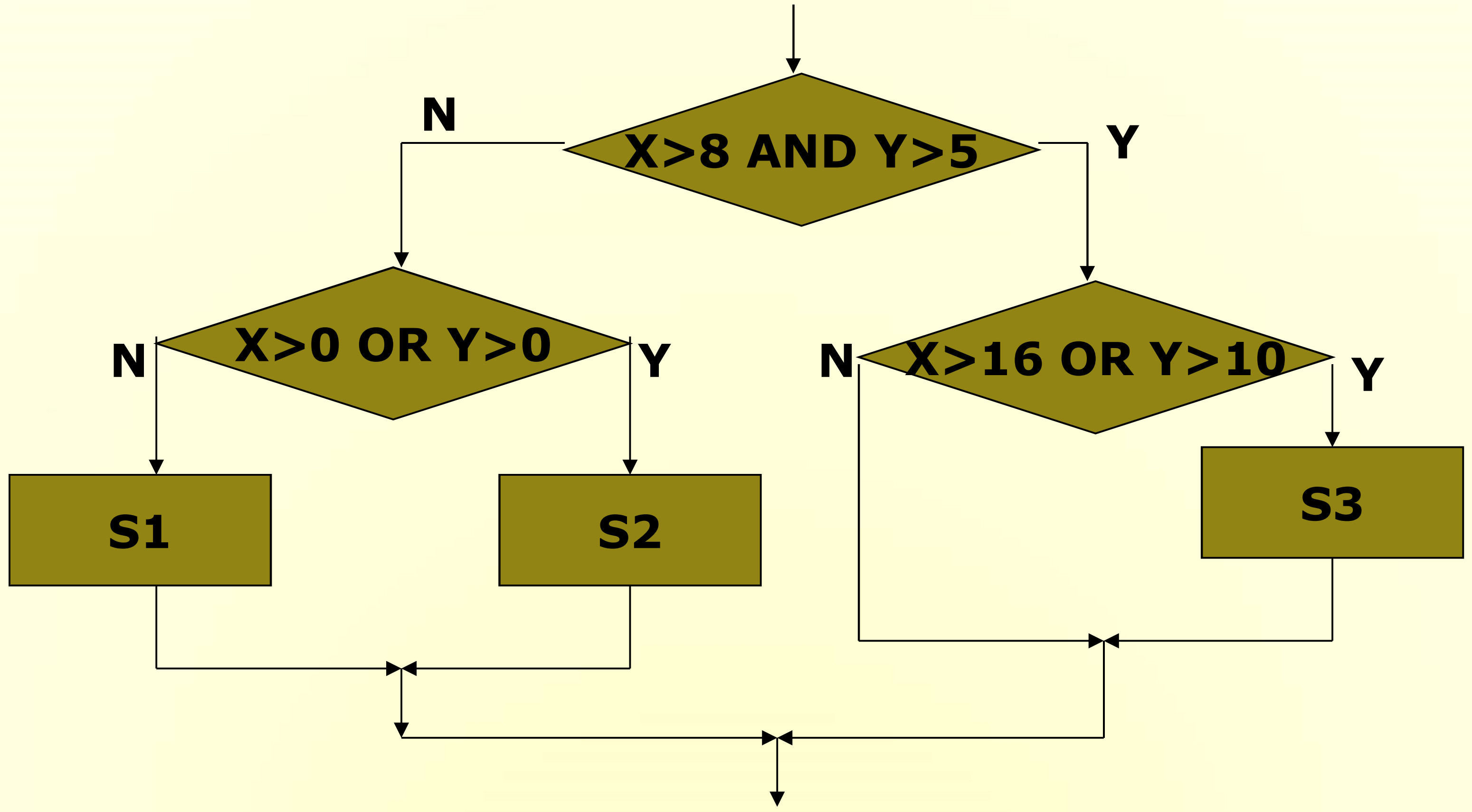
- White-box testing must follow several principles:
  - All independent path in a module must be implemented at least once. (Basis path testing)
  - All logic values require two test cases: true and false. (Logic coverage)
  - Inspection procedures of internal data structure, and ensuring the effectiveness of its structure.
    (Static Testing + Data Flow Testing)
  - Run all cycles within operational range.
    (Loop testing)

3.2 Logic Coverage

- Logic coverage
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Condition/Decision Coverage
  - Modified Condition/Decision Coverage
  - Condition Combination Coverage
  - Path Coverage
  - Complete Coverage

Exercise 5

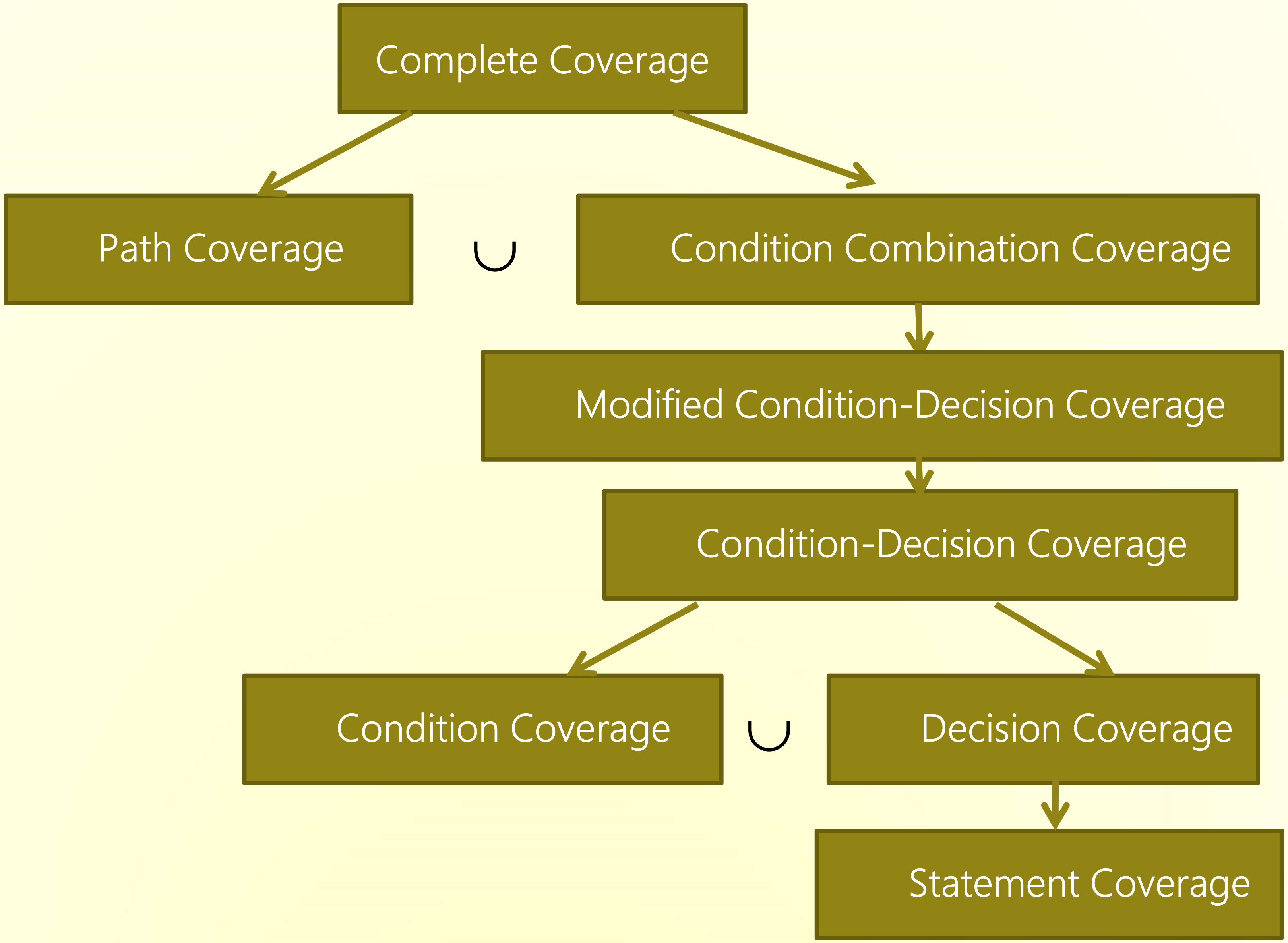
- Design test cases for the following flowchart to satisfy 8 types of coverage。(Note: X and Y are signed integers or 0)



Exercise 6

- Try to analyze the relationship among 8 different coverage strategies
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Condition/Decision Coverage
  - Modified Condition/Decision Coverage
  - Condition Combination Coverage
  - Path Coverage
  - Complete Coverage

Exercise 6


Complete Coverage
Path Coverage
∪
Condition Combination Coverage
Modified Condition-Decision Coverage
Condition-Decision Coverage
Condition Coverage
∪
Decision Coverage
Statement Coverage

Objectives

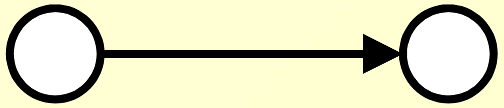
- In session 4, you will learn
  - Basic Concepts of White box testing
  - Logic Coverage
  - Control Flow Graph
  - Basis Path Testing
  - Loop Testing
  - Data Flow Testing

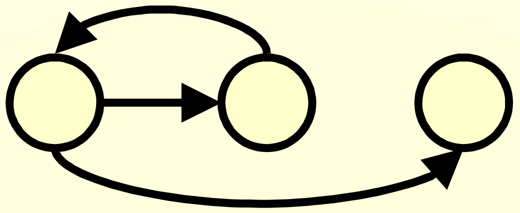
3.3 Control Flow Graph

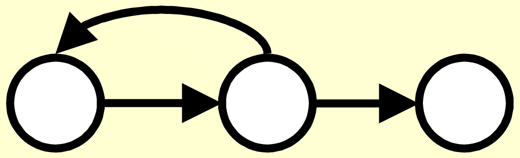
- Concept
  - During procedure design , in order to more prominent the control flow structure, the procedure can be simplified, the simplified graph is called control flow graph.   (vs. program flow graph)
  - On a flow graph:
    - Arrows called *edges* represent flow of control.
    - Circles called *nodes* represent one or more actions.
    - Areas bounded by edges and nodes called *regions*.
    - A *predicate node* is a node containing a condition.

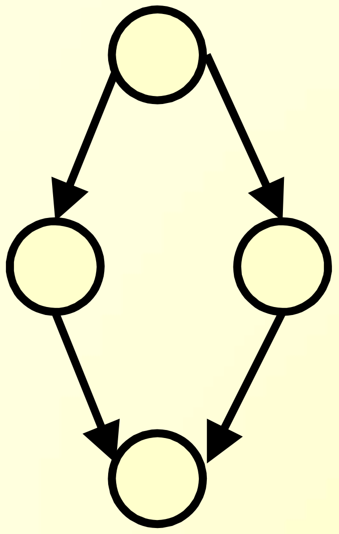
3.3 Control Flow Graph

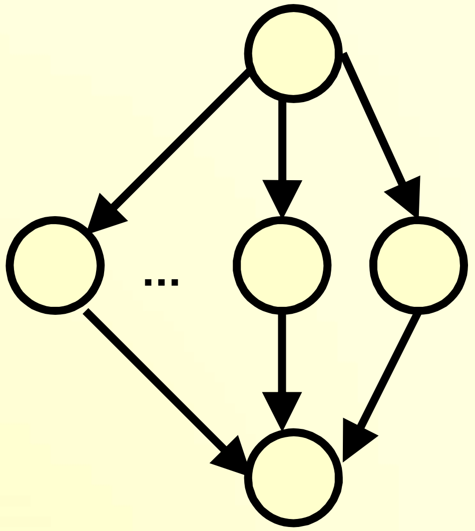
- Common control flow graph


Order statement


While statement

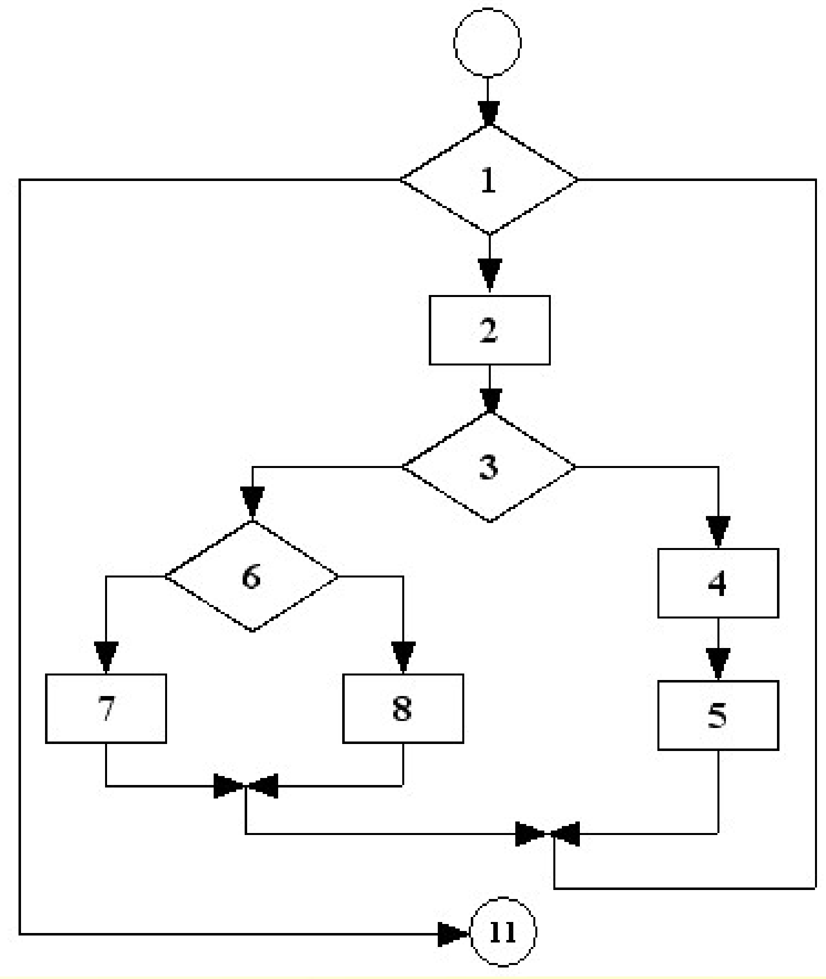

Until statement


IF statement

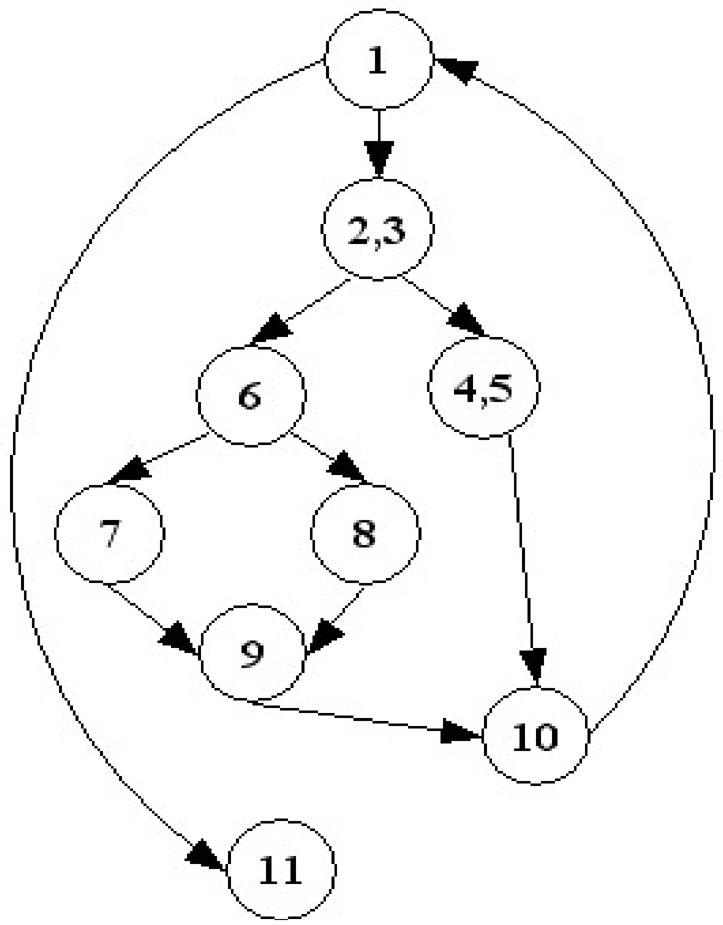

Case statement

3.3 Control Flow Graph

- Change a program flow chart into a control flow graph



（a）program flow chart



（b）control flow graph

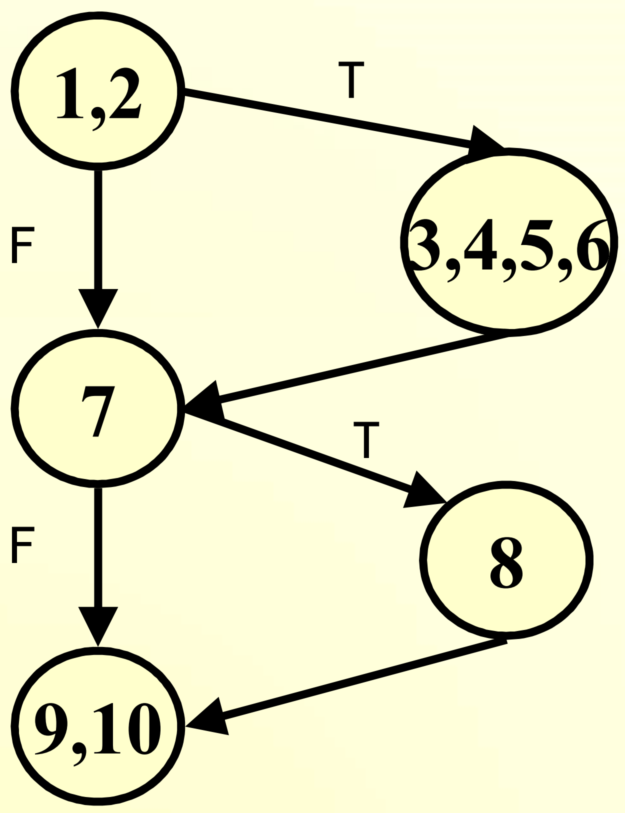
Exercise 1

- Draw the control flow graph of the program fragment below

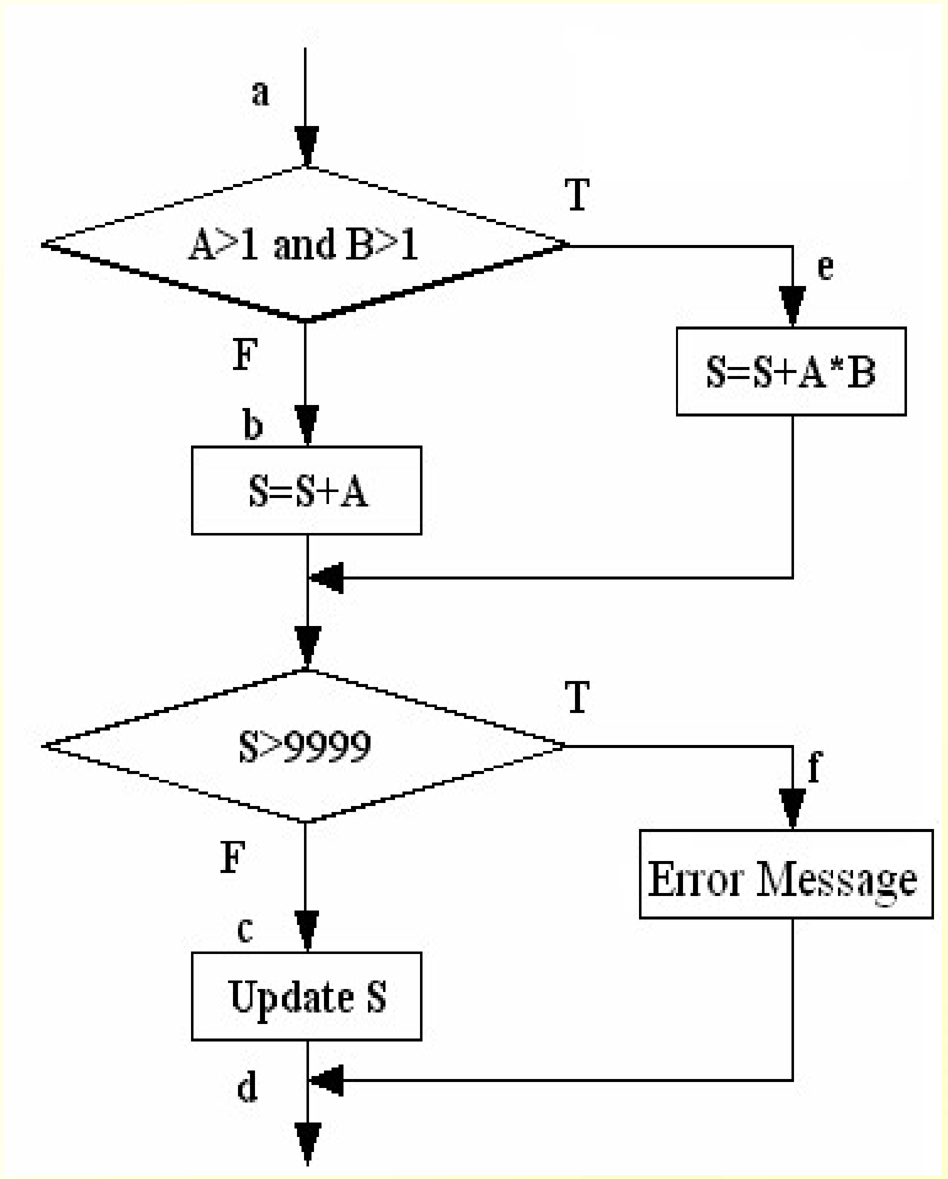
```
void DoWork(int x, int y, int z)
    {
1       int k=0,j=0;
2       if ((x>3)&&(z<10))
3        {
4               k=k*y-1;
5               j=sqrt(k);
6        }
7       if ((x==4)||(y>5))
8              j=x*y+10;
9       j=j%3;
10 }
```

Exercise 1

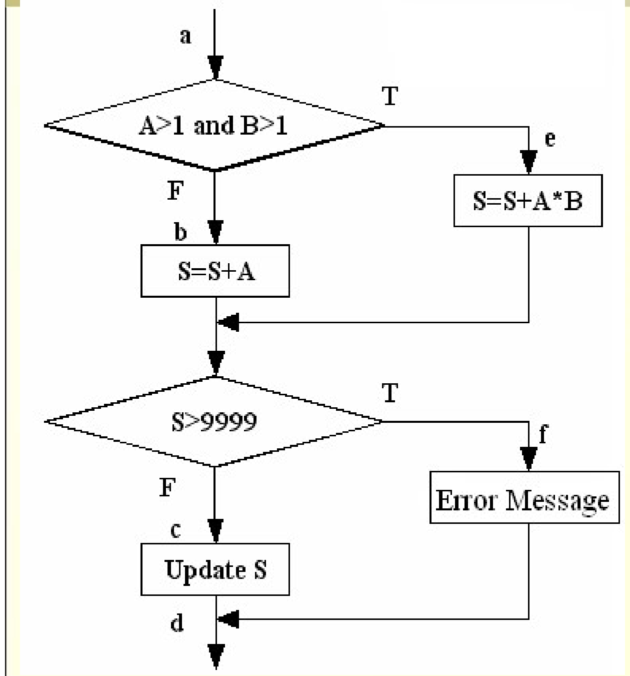
- Answer
  - Control flow graph

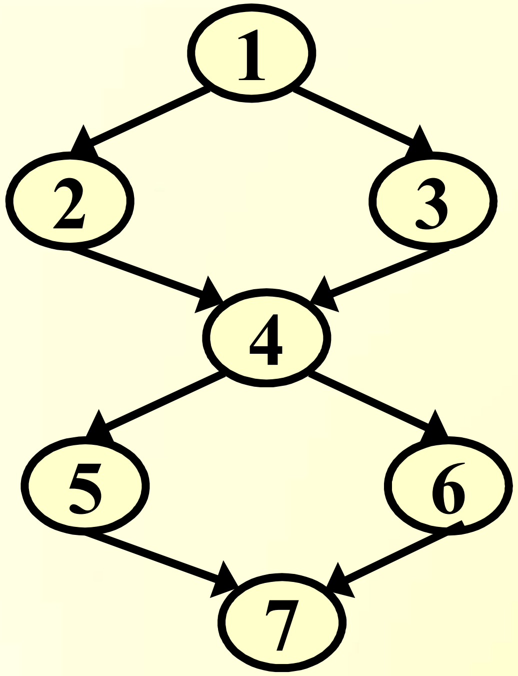

$$\begin{tikzcd}
1,2 \arrow[r, "T"] \arrow[d, "F"'] & 3,4,5,6 \arrow[dl] \\
7 \arrow[r, "T"] \arrow[d, "F"'] & 8 \arrow[dl] \\
9,10 &
\end{tikzcd}$$

Exercise 2



- **Requirement**
  - **Draw control flow graph**
  - **Use 8 types of logic coverage to design test case.**



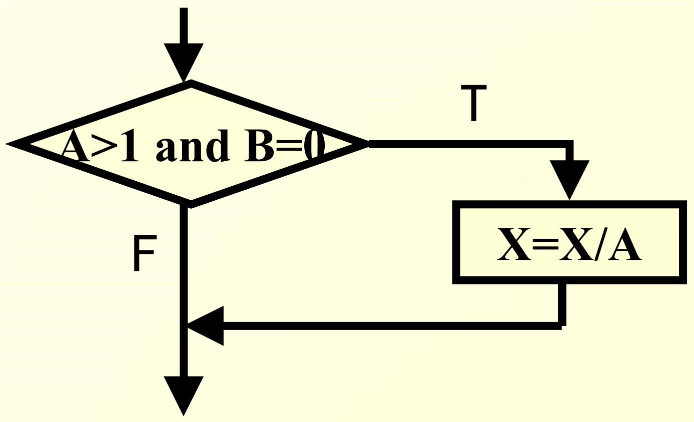


Exercise 2

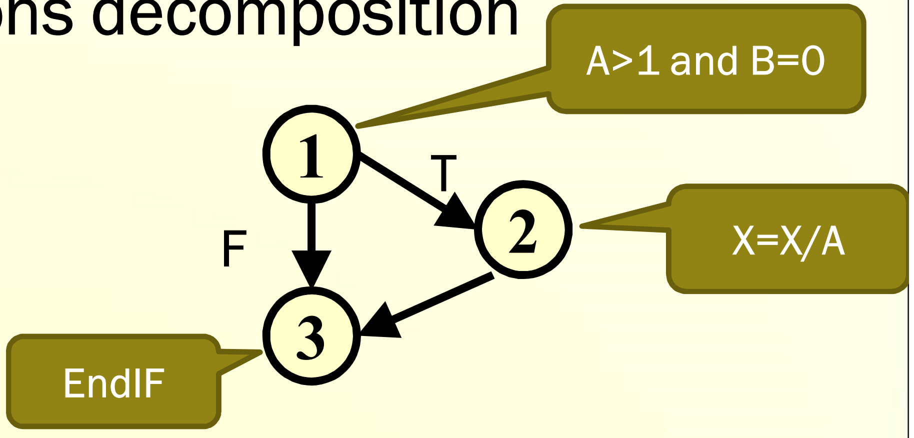
No.	Coverage type	Test case	Expected output
1	Statement coverage	A=2，B=2，S=9999	Error Message
2		A=1，B=1，S=9997	Update S
3	Decision coverage	A=2，B=2，S=9999 (TT)	Error Message
4		A=1，B=1，S=9997(FF)	Update S
5	Condition combination coverage	A=2，B=2，S=9997	Error Message
6		A=1，B=2，S=9999	Error Message
7		A=1，B=1，S=9997	Update S
8		A=2，B=1，S=9997	Update S
…	…	…	…

3.3 Control Flow Graph

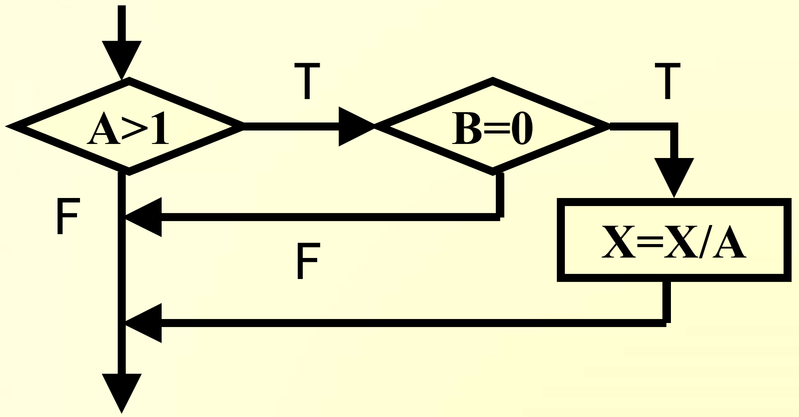
* Control flow graph of conditions decomposition



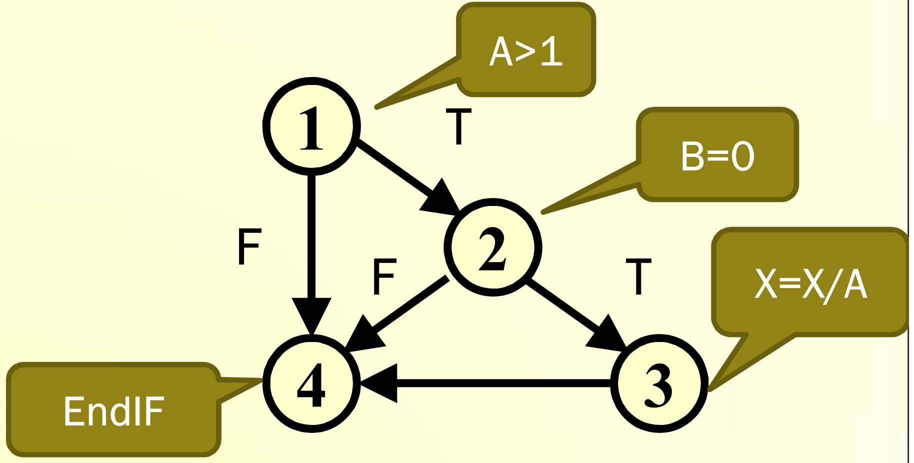
（1）Program flow chart



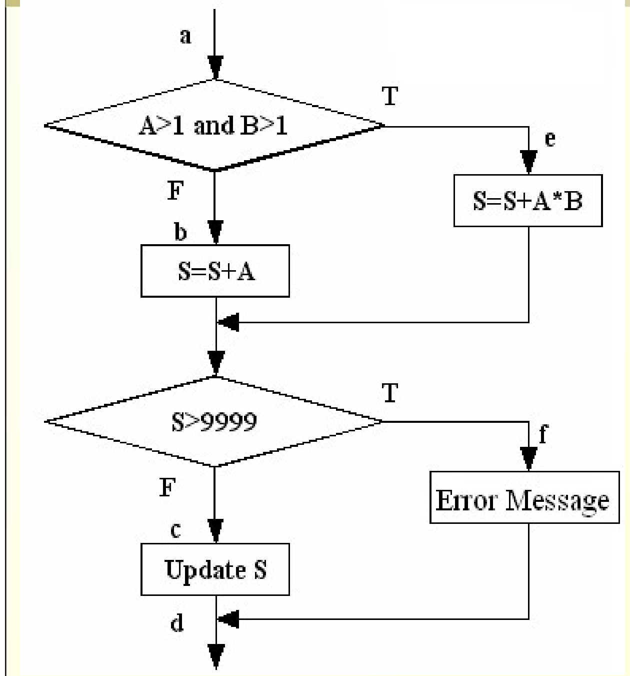
（2）corresponding control flow graph to program flow chart（1）



（3）Detailed program flow chart

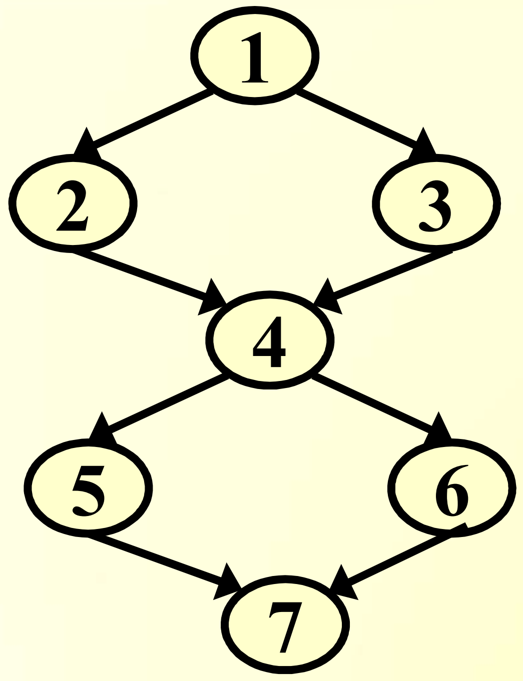


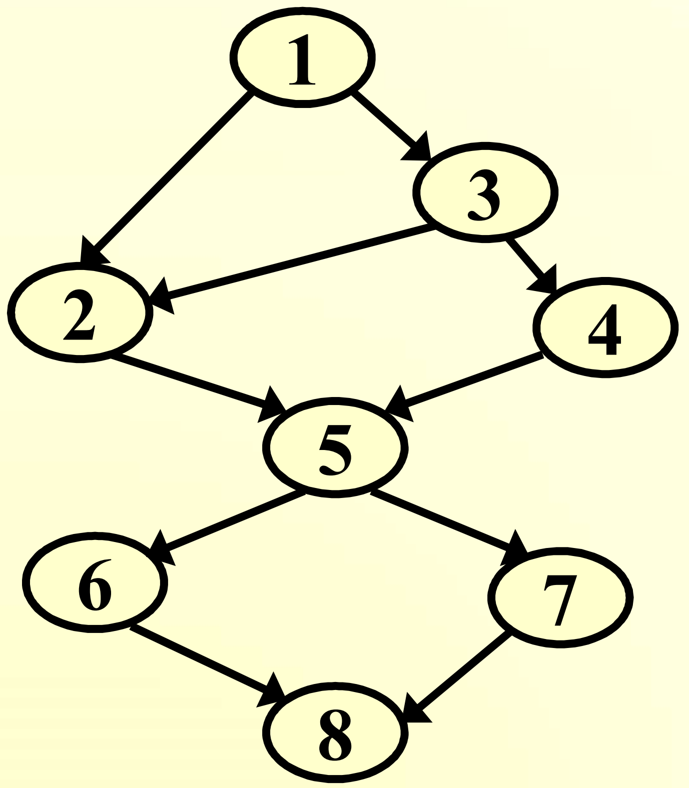
（4）corresponding control flow graph to program flow chart（3）



Exercise 2-continued

- Draw control flow graph of conditions decomposition





Exercise 3

```
void test(int iStudents, int iData[ ])
0  {
1    int x=0;
2    int y=0;
3    int z=0;
4    for(int loop=0; loop< iStudents;loop++)
5    {
6         int dNum= iData[loop];
7         if(dNum<60)
8            x++;
9         else
10        {
11          if(dNum>=60)
12             y++;
13          else
14             z++;
15         }
16     }
17  }
```

Exercise 3

$$\begin{tikzcd}
 & & 0\text{-}3 \arrow[d] & & & \\
 & & 4 \arrow[d] \arrow[dddddll, bend right] & & & \\
 & & 5\text{-}7 \arrow[dl] \arrow[drr] & & & \\
 & 8 \arrow[dddrrr] & & & 9\text{-}11 \arrow[dl] \arrow[dr] & \\
 & & & 12 \arrow[dr] & & 13\text{-}14 \arrow[dl] \\
 & & & & 15 \arrow[d] & \\
17 & & & & 16 \arrow[uuuuull, bend right] &
\end{tikzcd}$$

3.4 Basis Path Testing

- A testing mechanism proposed by McCabe.
- This method generates a set of linearly independent paths, which we called basis paths, from Control Flow Graph (CFG) and all the other paths can be expressed by them.

3.4 Basis Path Testing

- Why we need basis path testing?
  - Exhaustive path testing is usually impossible.
  - Aim is to derive a logical complexity measure of a procedural design and use this as a guide for defining a basic set of execution paths.
  - Test cases which exercise basic set will execute every statement at least once.

3.4 Basis Path Testing

- Basis path testing is a hybrid between path testing and branch testing:
  - Path Testing: Testing designed to execute all or selected paths through a program.
  - Branch Testing: Testing designed to execute each outcome of each decision point in a program.
  - Basis Path Testing: Testing that fulfills the requirements of branch testing & also tests all of the independent paths that could be used to construct any arbitrary path through the program.

3.4 Basis Path Testing

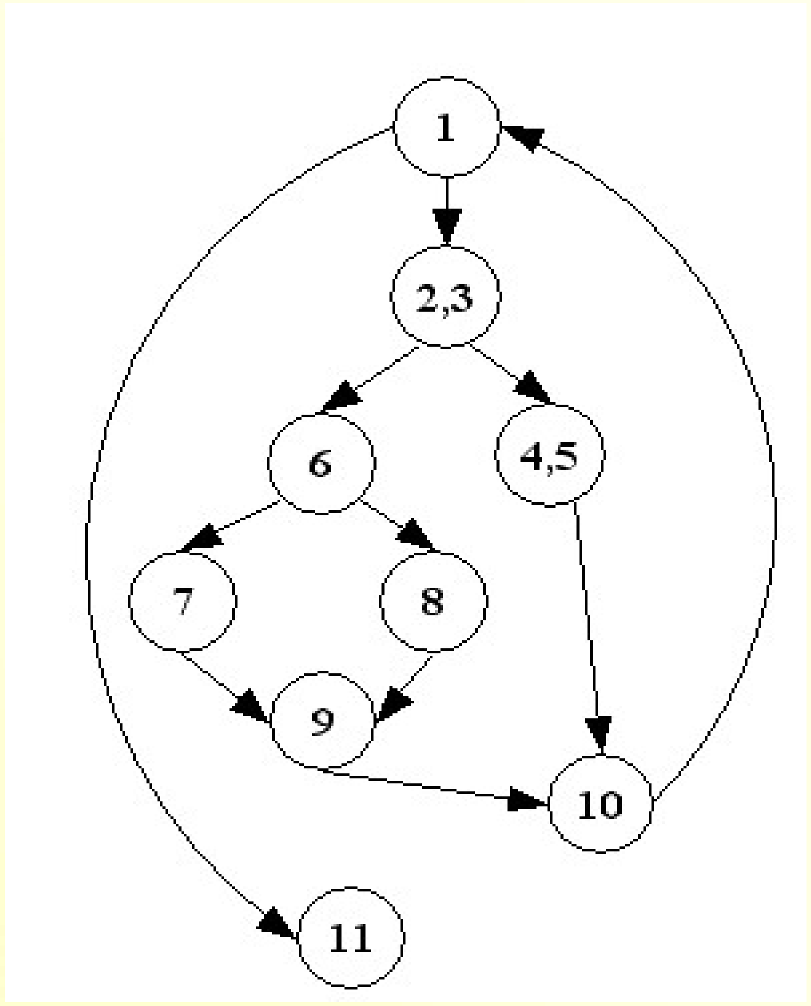
- How to design test cases for basis path testing
  - Using the design or code, draw the corresponding control flow graph.
  - Determine the cyclomatic complexity of the flow graph.
  - Determine a basis set of independent paths.
  - Prepare test cases that will force execution of each path in the basis set.

3.4 Basis Path Testing

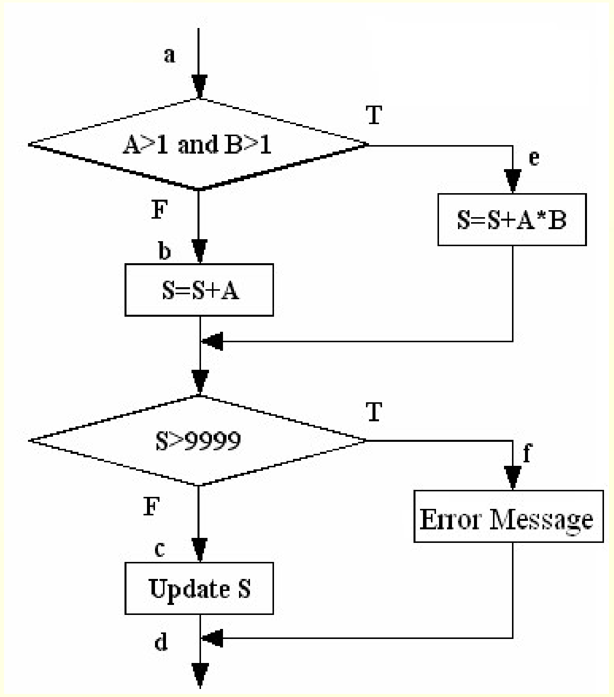
- Cyclomatic Complexity （*环路复杂度/圈复杂度*）
  - It gives a quantitative measure of the logical complexity.
  - This value gives the number of independent paths in the basis set, and an upper bound for the number of tests to ensure that each statement and both sides of every decision/condition is executed at least once.
  - An independent path is any path through a program that introduces at least one new set of processing statements (i.e., a new node) or a new condition (i.e., a new edge)

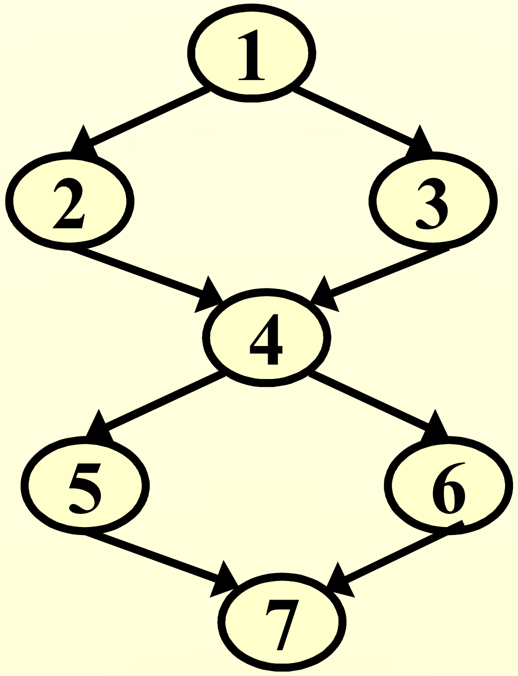
3.4 Basis Path Testing

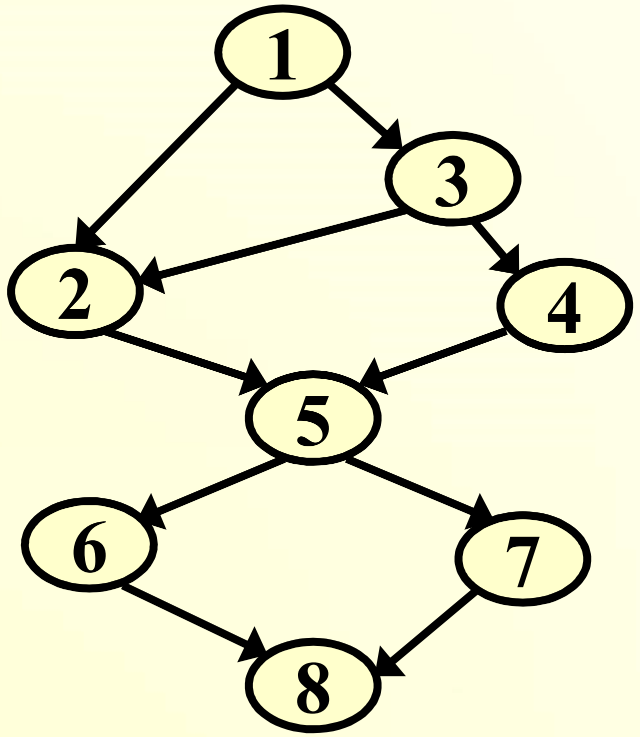
- Cyclomatic Complexity
  - #Edges - #Nodes + 2
  - #Predicate Nodes + 1
  - # closed regions + 1



3.4 Basis Path Testing



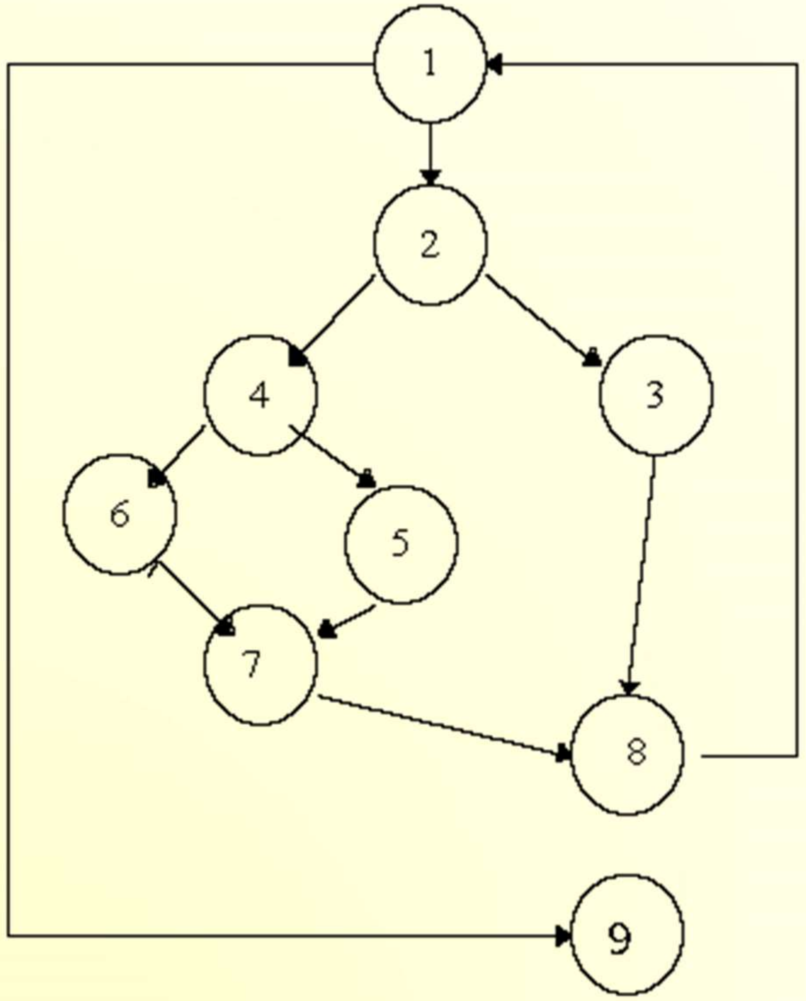




- Different Cyclomatic Complexity
  - Statement coverage
  - Decision coverage / Condition coverage

3.4 Basis Path Testing

- Begin at the module's entry point, take the leftmost path through the module to its exit. Return to the beginning and vary the branching condition.
- Like depth-first searching.
- Short path (new node/edge)
- Independent Paths:
  - P1: 1, 9
  - P2: 1, 2, 4, 6, 7, 8, 1, 9
  - P3: 1, 2, 4, 5, 7, 8, 1, 9
  - P4: 1, 2, 3, 8, 1, 9



3.4 Basis Path Testing

- Summary
  - Using the design or code, draw the corresponding flow graph.
  - Determine the cyclomatic complexity of the flow graph.
  - Determine a basis set of independent paths.
  - Prepare test cases that will force execution of each path in the basis set.

Exercise 3

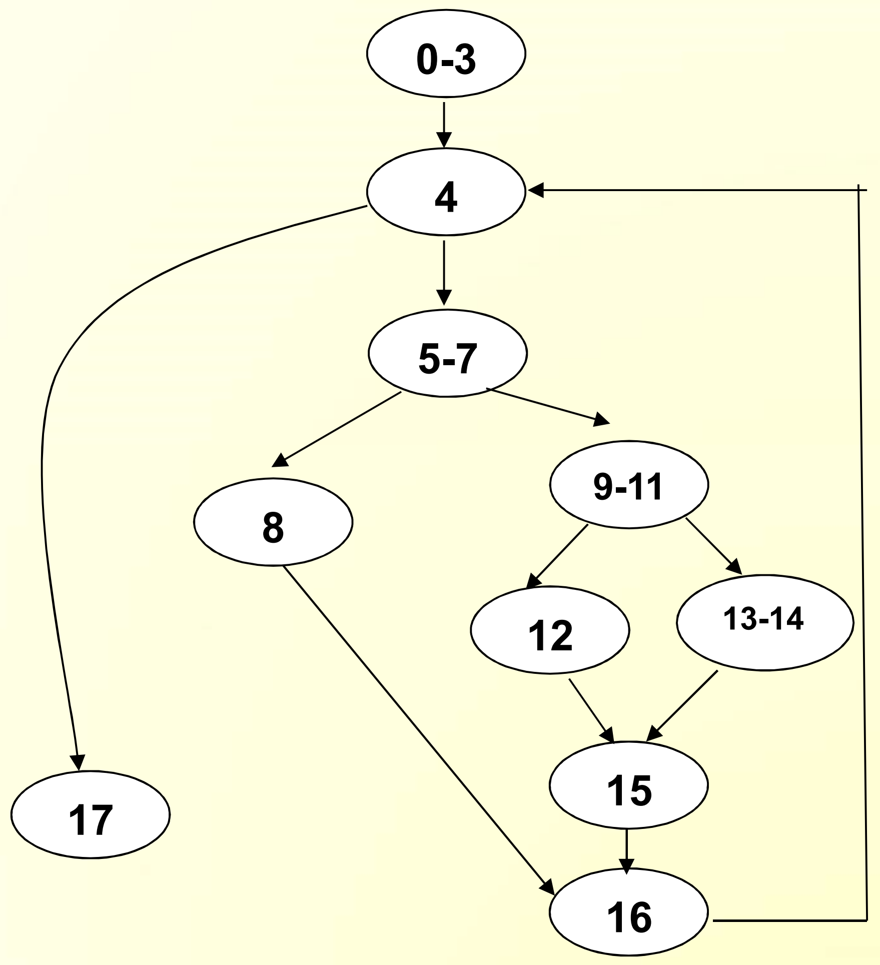
```
void test(int iStudents, int iData[ ])
0  {
1    int x=0;
2    int y=0;
3    int z=0;
4    for(int loop=0; loop< iStudents; loop++)
5    {
6      int dNum= iData[loop];
7      if(dNum<60)
8          x++;
9      else
10       {
11         if(dNum>=60)
12            y++;
13         else
14            z++;
15       }
16   }
17 }
```

- Requirements
  - Draw the control flow graph.
  - Calculate the cyclomatic complexity.
  - Derive the basis set of independent paths.
  - Design the test cases.

Exercise 3

$$\begin{tikzcd}
 & & \text{0-3} \arrow[d] & & \\
 & & 4 \arrow[d] \arrow[ddddll] & & \\
 & & \text{5-7} \arrow[dl] \arrow[dr] & & \\
 & 8 \arrow[dddrr] & & \text{9-11} \arrow[d] \arrow[dr] & \\
 & & & 12 \arrow[d] & \text{13-14} \arrow[dl] \\
17 & & & 15 \arrow[d] & \\
 & & & 16 \arrow[uuuuul, bend right] &
\end{tikzcd}$$

Exercise 3



- Cyclomatic complexity
  - 3+1=4

- 3-4-17
- 3-4-7-8-16-4-17
- 3-4-7-11-12-15-16-4-17
- 3-4-7-11-14-15-16-4-17

Exercise 3

2, [30,82]

Path	Input int iStudents, int iData[ ]
3-4-17	-1, [2,3]
3-4-7-8-16-4-17	1, [30]
3-4-7-11-12-15-16-4-17	1, [80]
3-4-7-11-14-15-16-4-17	Null

3, [30,40,50]

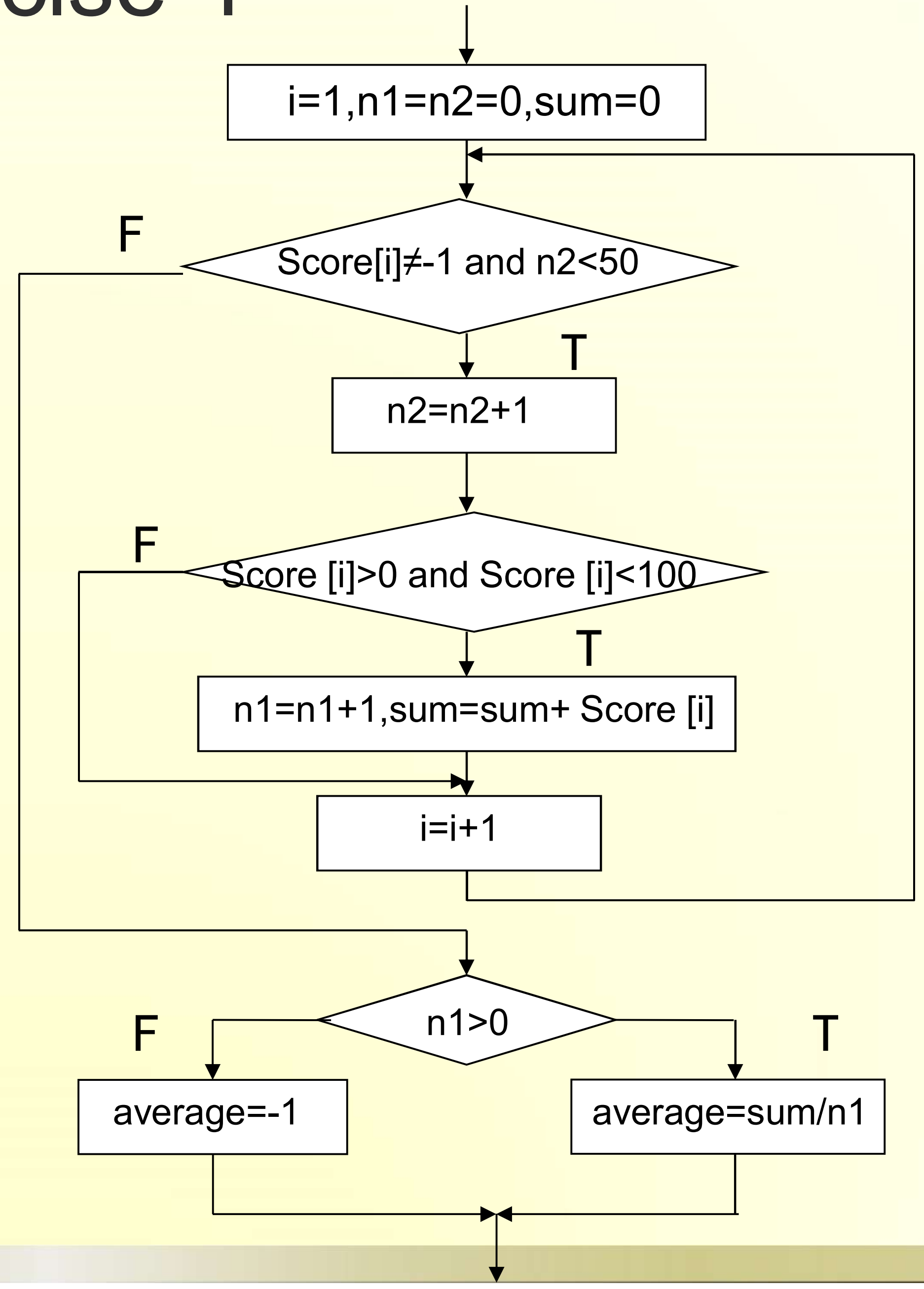
3, [80,82,85]

Dead code / Infeasible path

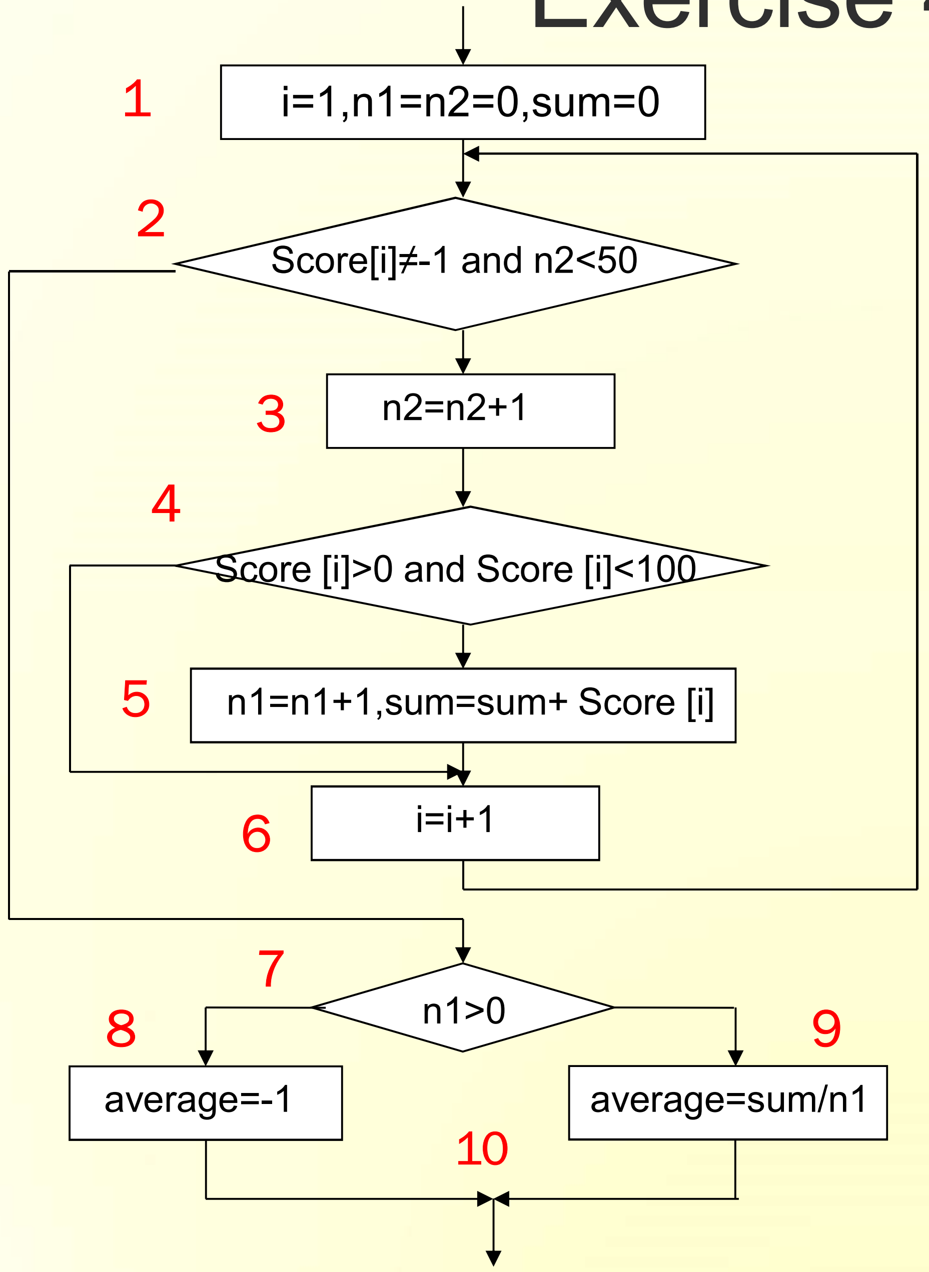
Exercise 4

- Inputs: Score[i]
- Outputs: n2, sum, average

- No conditions decomposition
- Conditions decomposition



Exercise 4-- Answer 1

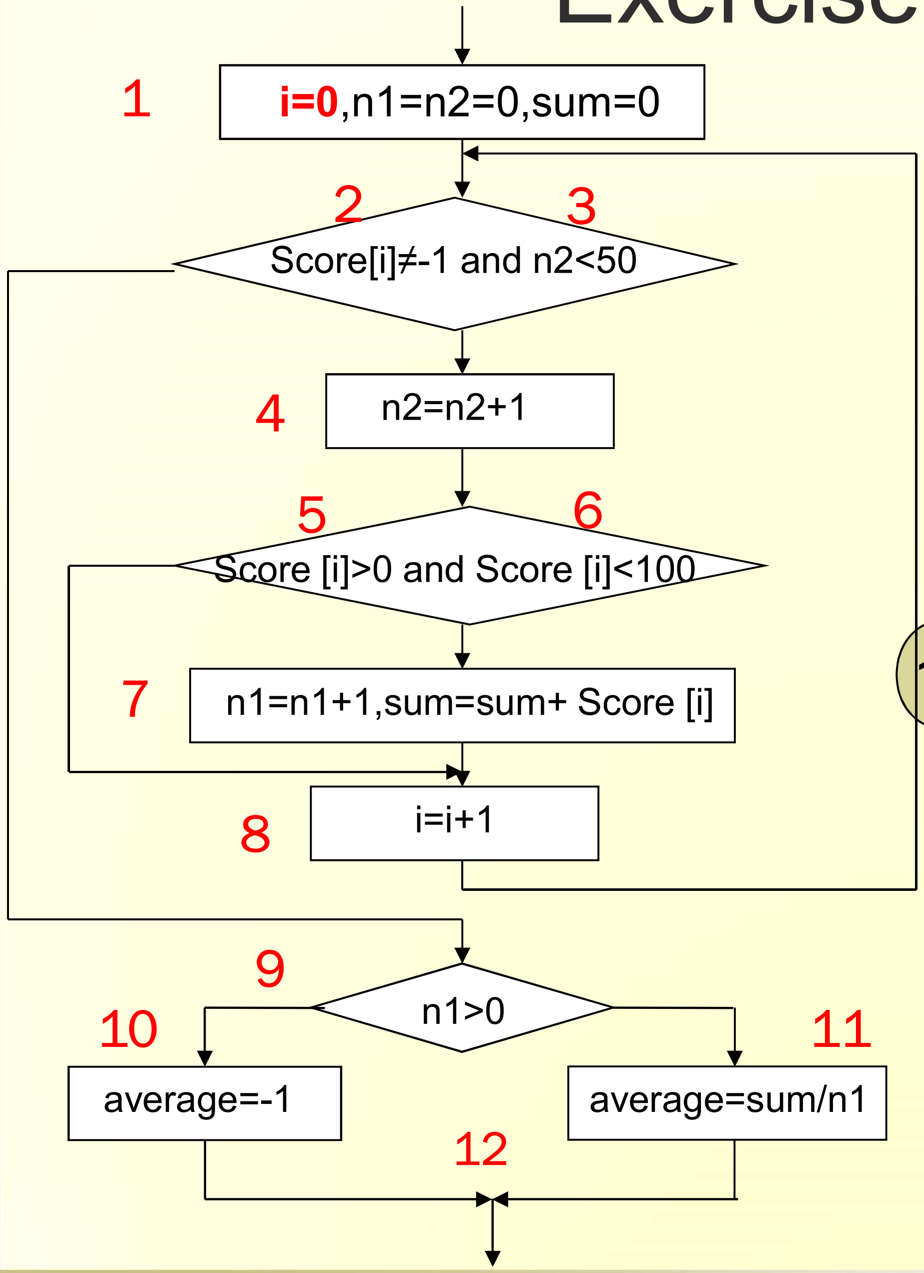


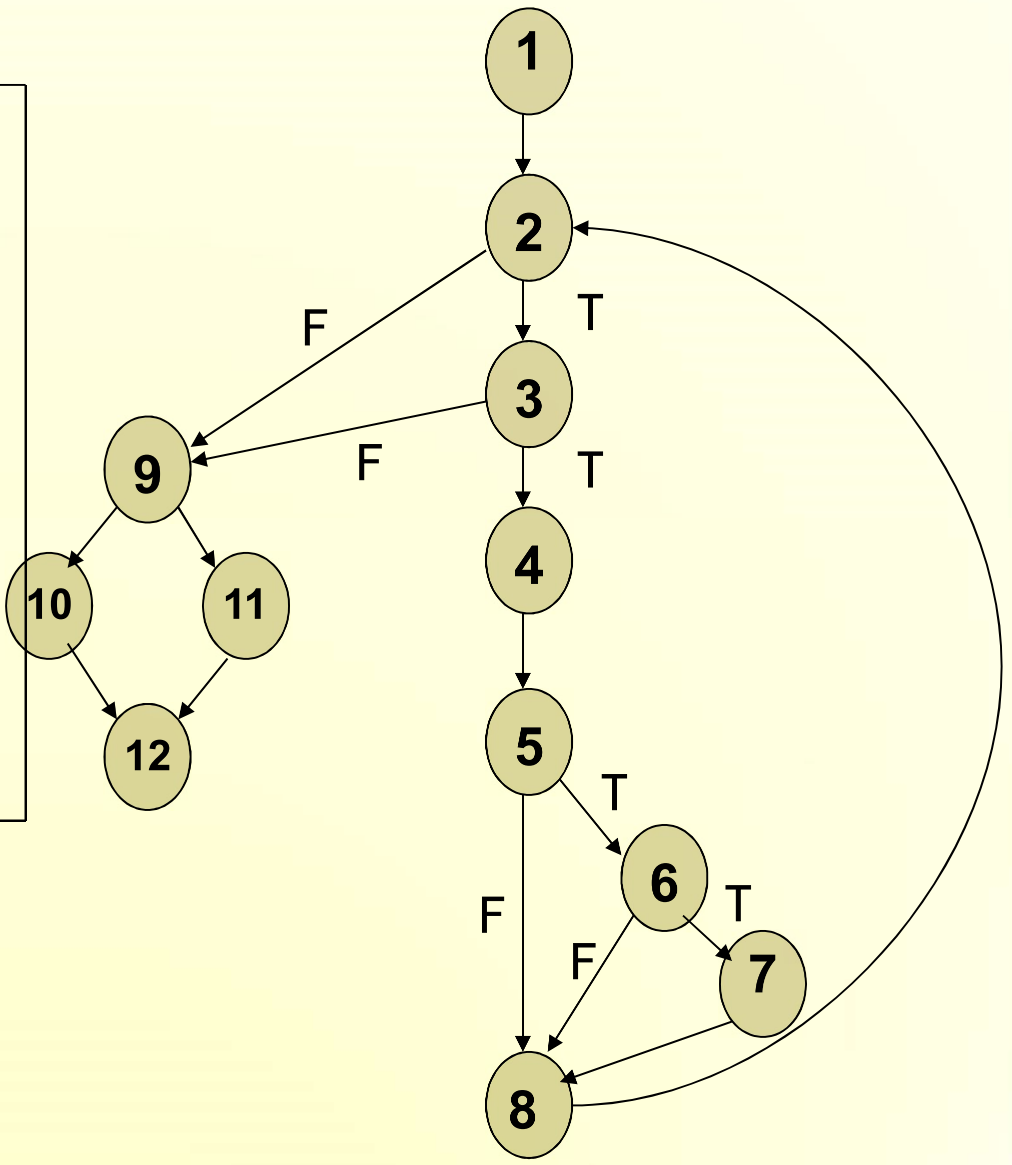
$$\begin{tikzcd}
1 \arrow[d] & & & \\
2 \arrow[dr] \arrow[dddddr, bend right] & & & \\
& 3 \arrow[dr] & & \\
& & 4 \arrow[dr] \arrow[dd] & \\
& & & 5 \arrow[dl] \\
& & 6 \arrow[uuuull, bend right] & \\
& 7 \arrow[dl] \arrow[dr] & & \\
8 \arrow[dr] & & 9 \arrow[dl] & \\
& 10 & &
\end{tikzcd}$$

Exercise 4

- For Answer 1, derive the basis paths :
  - Path 1: 1-2-7-8-10
  - Path 2 : 1-2-7-9-10
  - Path 3 : 1-2-3-4-6-2-7-8-10
  - Path 4 : 1-2-3-4-5-6-2-7-9-10

Exercise 4-- Answer 2





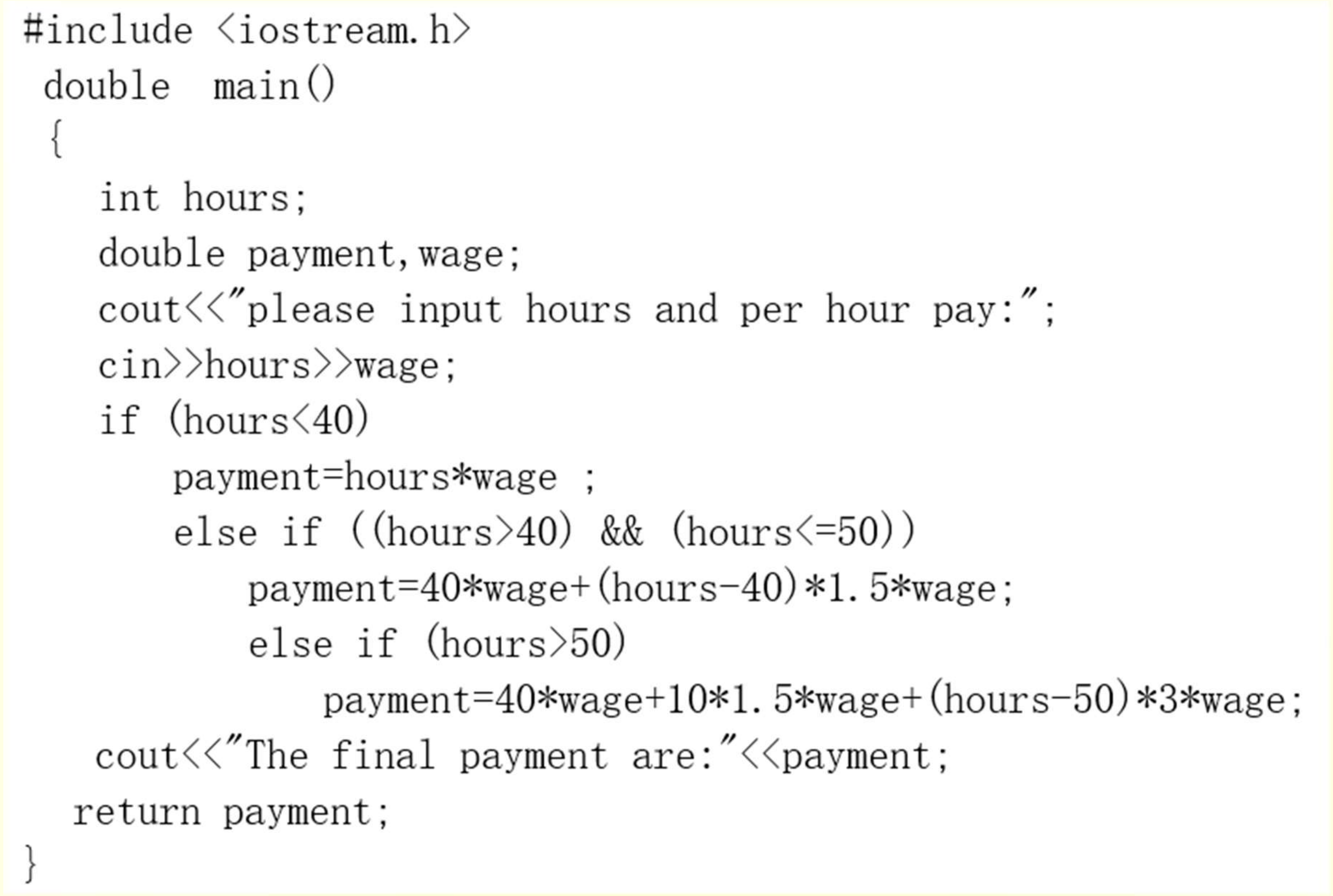
Exercise 4

- For Answer 2, derive the basis paths :
  - Path 1: 1-2-9-10-12
  - Path 2 : 1-2-9-11-12
  - Path 3 : 1-2-3-9-10-12
  - Path 4 : 1-2-3-4-5-8-2-9-10-12
  - Path 5 : 1-2-3-4-5-6-8-2-9-10-12
  - Path 6 : 1-2-3-4-5-6-7-8-2-9-11-12

Exercise 3/4

- Design test cases.

QUIZ-1 Logic Coverage



```
#include <iostream.h>
 double  main()
 {
    int hours;
    double payment,wage;
    cout<<"please input hours and per hour pay:";
    cin>>hours>>wage;
    if (hours<40)
        payment=hours*wage ;
        else if ((hours>40) && (hours<=50))
            payment=40*wage+(hours-40)*1.5*wage;
            else if (hours>50)
                payment=40*wage+10*1.5*wage+(hours-50)*3*wage;
    cout<<"The final payment are:"<<payment;
   return payment;
}
```

QUIZ-2 Basis Path Testing

```
void insertionSort(int numbers[ ], int array_size)
0{
1    int i, j, index;
2    for (i=1; i < array_size; i++)
3    {
4          index = numbers[i];
5          j = i;
6         while ((j > 0) && (numbers[j-1] > index))
7         {    numbers[j] = numbers[j-1];
8              j = j - 1;
9         }
10         numbers[j] = index;
11   }
12}
```

Objectives

- In session 3-4, you have learned
  - Basic Concepts of White box testing
  - Logic Coverage
  - Control Flow Graph
  - Basis Path Testing
  - Loop Testing
  - Data Flow Testing